LE BASI DELLA CRITTOGRAFIA

BIOGRAFIA

Rosario Turco è un ingegnere elettronico, laureato all'Università Federico II di Napoli, che lavora dal 1990 in società del gruppo Telecom Italia.

Le sue competenze professionali sono in ambito delle archittetture hw/sw Object Oriented (OOA/OOP, AOP, SOA, Virtualization) e in generale "Java 2 Enterprise Edition". Ha lavorato molti anni nella progettazione e sviluppo di sistemi informatici su piattaforme Windows/Unix/Linux e con linguaggi Java, C, C++.

Negli ultimi anni si sta particolarmente interessando alla crittografia e alla Teoria dei Numeri.

INTRODUZIONE

Nella crittografia fanno la "parte da leone" la Matematica con la Teoria dei Numeri e l'Informatica con l'Analisi computazionale e di complessità. Per poter affrontare problematiche di crittografia, nel seguito introduciamo, senza pretesa di rigore, con la giusta sintesi e in versione non noiosa, quella parte minima di matematica che possa permettere di comprendere il tutto anche ai non addetti.

ARITMETICA MODULARE

L'aritmetica modulare è basata sull'operazione modulo; ovvero quella operazione che definiti X come il dividendo, m come il divisore, Q come quoziente ed R come resto fornisce il resto della divisione.

In formula è:

X (mod m) = R

oppure più sinteticamente:

X @m = R

Esempi

10 @4 = 2
21 @45 = 21
57 @57 = 0
12 @1 = 0
37 @36 = 1

Proprietà del modulo:

- E' sempre R < m
- Tutti i possibili R sono una quantità pari ad m e con un valore compreso tra 0 e m-1
- Se X < m, X @m = X o R=X
- Se X = m, X @m = 0 o R = 0
- Se m = 1, X @m = 0
- X @X-1 = 1
- essendo i risultati comunque dei Resti, se il risultato è maggiore o uguale a m occorre proseguire a dividerlo per m (d'altra parte come si fa nella normale

divisione ovvero la divisione non è terminata perché il resto è maggiore o uguale al divisore m)

Proprietà notevole del modulo di una somma
(X + Y) @m = X @m + Y @m ovvero "Il Resto di una somma è la somma dei resti"

Proprietà notevole del modulo di un prodotto
(X Y) @m = X @m Y @m ovvero "Il Resto di un prodotto è il prodotto dei resti"

Proprietà notevole del modulo di un quadrato
X^2 @m = (X *X) @m = X @m X @m = R2 ovvero "Il Resto di un quadrato è il quadrato del resto"

Con la proprietà del quadrato è possibile calcolare il resto di divisioni di numeri con un numero elevato di cifre (divisioni impossibili).

Esempi:
17 @5 = 2 = (12 + 5) @5 = 12 @5 + 5 @5 = 2 + 0 = 2
24 @5 = 4 = (6 . 4) @5 = 6 @5 4 @5   = 1 . 4 = 4
252 @11 = 625 @11 = 9 = (25 @11)2 = 3 2 = 9
7-9 @20 = 18@20
15+9@20 = 4 @20

DIVISIONI IMPOSSIBILI

Un esempio di divisione impossibile è il calcolo seguente:

441^{1183} @2867

E' impossibile perché un calcolatore sarebbe perduto se non sapesse una particolare proprietà come il "*resto di un quadrato*".

Esiste un algoritmo generalizzabile e implementabile, tale da poter dire rapidamente che il risultato è 2515.

L'algoritmo consiste in:"*Trasformare il numero in una potenza di 2 più vicina ovvero uguale o inferiore; sommare ad esso le potenze di 2, scartando però quelle potenze di 2 che farebbero superare la cifra iniziale*".

Nell'esempio precedente è:
1183 = 1024 + (scarto 512) + (scarto 256) + 128 + (scarto 64) + (scarto 32) + 16 + 8 + 4 + 2 + 1 = 2 10 + 2 7 + 2 4 + 2 3 + 2 2 + 2 1 + 2 0

Con la "*proprietà della potenza del resto*" e la proprietà 7) si ottiene:

441^{1183} @2867 = (441 * 441^2 * 441^4 * 441^8 * 441^{16} * 441^{128} * 441^{1024}) @2867 =
441 @2867 * 441^2 @2867 * 441^4 @2867 * 441^8 @2867 * 441^{16} @2867 * 441^{128} @2867 *
441^{1024} @2867

Ora è:

$441 @2867 = 441$
$441^2 @2867 = 194481 = 194181 @2867 = 2392$
$441^4 @2867 = 2392^2 = 5721664 = 5721664 @2867 = 1999$
$441^8 @2867 = 1999^2 = 3996001 = 3996001 @2867 = 2270$
$441^{16} @2867 = 2270^2 = 5152900 = 5152900 @2867 = 901$
$441^{32} @2867 = 901^2 = 811801 = 811801 @2867 = 440$
$441^{64} @2867 = 440^2 = 193600 = 193600 @2867 = 1511$
$441^{128} @2867 = 1511^2 = 2283121 = 2283121 @2867 = 989$
$441^{256} @2867 = 989^2 = 978121 = 978121 @2867 = 474$
$441^{512} @2867 = 474^2 = 224676 = 224676 @2867 = 1050$
$441^{1024} @2867 = 1050^2 = 1102500 = 1102500 @2867 = 1572$

Per cui è:
(441 * 2392 * 1999 * 2270 * 901 * 989 * 1572) @2867 = 2515

FUNZIONI MODULARI

Si dice *Funzione modulare* una funzione del tipo:

$$C = EP @m$$

Dove: E ed m sono costanti numeriche intere, C è la variabile dipendente e P la variabile indipendente. C è un resto. Poiché cercheremo anche una funzione modulare inversa P, anche P è un resto. In particolare C e P sono interi compresi tra 0 e m-1.

Si dimostra che **"***la funzione C = EP @m è invertibile se e solo se E ed m sono primi fra loro, ovvero non hanno fattori in comune***"**.

Esempio:

$$\mathbf{C = 4P @7}$$

P variabile Indipendente	0	1	2	3	4	5	6
C	0	4	1	5	2	6	3

La tabella mostra come la funzione modulo fornisce valori disordinati.

Si dimostra che la *funzione modulo inversa* è:

$$\mathbf{P = 2C} @7$$

Infatti inserendo i valori di C in una tabellina 0, 4 , 1, 5 , 2 , 6 ,3 si ritrovano quelli di P di prima.

FUNZIONI MODULARI INVERSE

"*Se E ed m sono primi fra loro la funzione C = EP @m è invertibile in P= DC @m***"**

Sappiamo in generale che:

$$X @m = R$$

Per cui moltiplicando entrambi i membri per un intero D si ottiene:

DX @m = DR = DR@m anche per la proprietà 7).

Allo stesso si può giungere sommando per una stessa quantità D:

$$(D+X) @m = (D+R)@m$$ anche per la proprietà 7).

Per cui diventano giustificati i passaggi:

$$DC @m = DEP @m = P DE @m$$

Ora per trovare P dobbiamo fare in modo che sia DE @m = 1 in modo che P = DC@m

D però esiste solo se E e m sono primi fra loro; in tal modo D è univoco.

Esempio:
C=4P @7   E=4 m=7 primi fra loro
Dobbiamo fare in modo che sia DE @m = 1 ovvero che 4D @7 = 1, quindi D=2; per cui è:
P=2C@7

Questo era un esempio semplice.

Un esempio complicato di ricerca della funzione modulo inversa è il seguente:
C=11P @10800

Occorre trovare un DE @m = 1 ovvero:
11D@10800 = 1, ovvero dobbiamo fare in modo che R = 1

Qual è l'algoritmo per arrivarci in breve?

ALGORITMO DELLA DIVISIONE CON ARRESTO AL VALORE INTERO

Indichiamo con Q il risultato della divisione tra DE e m, ed R=1 per cui è:

$$DE = m * Q + 1$$

Ora dobbiamo arrestare l'algoritmo solo quando come risultato otteniamo un intero da

$$D = m * Q +1 / E.$$

Per l'esempio precedente dopo 6 iterazioni si ottiene D=5981. Per cui è: P=5981C@10800

FUNZIONI MODULARI ESPONENZIALI

Si chiama *funzione di Eulero N(m) di un numero m*, il numero di naturali, 1 incluso, compresi tra 1 e m-1 che non hanno fattori in comune con m.

Esempio:
N(9) =6   ovvero i numeri 1,2,4,5,7,8
N(12)=4   ovvero i numeri 1,5,7,11
N(18)=6   ovvero i numeri 1,5,7,11,13,17

Proprietà
Se m=p*q dove p e q sono primi
N(m) = N(p * q) = (p-1)(q-1)

La proprietà è estendibile anche al prodotto di x numeri primi.

Nella crittografia, la funzione di Eulero è molto importante data la sua semplice determinazione anche in presenza di numeri primi molto grandi.

Una funzione modulare esponenziale è del tipo:

C = PE @m

La funzione modulare esponenziale è invertibile se E ed N(m) sono primi fra loro.

FUNZIONI MODULARI ESPONENZIALI INVERTIBILI

In generale è:

X @m = R

Se D è un intero, è ancora:

XD @m = RD @m

Infatti è:

XD @m = (X*X*X*...*X) @m = X @m * X @m * X @m * ... * X @m = R*R*R*...*R =RD @m

Ora nella Teoria dei Numeri esiste un teorema che permette di scrivere la seguente espressione:

C = PE @m = C = PE(@N(m)) @m

Da qui è anche:

CD = PDE(@N(m)) @m

Per invertire una funzione modulare esponenziale deve essere DE(@N(m)) = 1, così si ottiene P = CD @m; ma D esiste se e solo se E ed N(m) sono primi fra loro.

Esempio
Per trovare N(m) sfruttiamo le proprietà della funzione di Eulero viste precedentemente. Ipotizziamo che m sia prodotto di due numeri primi: m=47*61
N(m)=N(47*61)=N(2867)=46*60=2760

La scomposizione in fattori di m=23 * 3 * 5 * 23. Scegliamo allora E primo rispetto ad m, es: E=7*132=1183.

La funzione modulare esponenziale che ne deriva è:

C=P1183 @2867

Per trovare la funzione modulare esponenziale inversa deve essere DE(@N(m)) = 1, ovvero che D1183 @2867 = 1.

Se si utilizza la tecnica dell'algoritmo della divisione con arresto al valore intero si ottiene dopo 3 iterazioni D=7. Quindi P = C7 @m

CONTROLLO DELLA PRIMALITA' COL PICCOLO TEOREMA DI FERMAT

Un numero si dice primo se è divisibile per 1 e per sé stesso; altrimenti è composto, ovvero scomponibile in fattori.

Il 2 è un numero primo ed è l'unico numero primo pari; mentre gli altri sono dispari. Il controllo della primalità serve a stabilire se un numero è primo o composto.

Un veloce algoritmo che nel 90% dei casi individua un numero primo (probabilmente primo) anche con un numero elevato di cifre è quello che segue.

"Piccolo Teorema di Fermat": "Se p è un numero primo, allora per ogni numero naturale b appartenente all'intervallo aperto (0,p), è:

$$b^p @ p = b$$

Altrimenti il numero è composto".

I numeri p=2,3 e 5 sono primi difatti perché:

p=2: $1^2@2=1$
p=3: $1^3@3=1$; $2^3@3=2$
p=5: $1^5@5=1$; $2^5@5=2$; $3^5@5=3$; $4^5@5=4$

Mentre è composto:
p=4: $1^4@4=1$; $2^4@4=0$; $3^4@4=1$;

Attenzione: purtroppo il Piccolo Teorema di Fermat (PTF) non è sempre valido! Ad esempio secondo tale teorema i numeri composti 561=3*11*17, 1729=7*13*19 si comportano come se fossero primi per tutti i valori dell'intervallo aperto (0,p).

I numeri composti che comunque soddisfano il PTF e che si comportano come numeri primi si dicono <u>*numeri di Carmichel*</u>.

Altri numeri composti rispettano il PTF ma non per tutti i valori dell'intervallo aperto (0,p). In questo secondo caso si parla di <u>*numeri pseudoprimi*</u>. I valori che rispettano il teorema sono chiamati <u>basi</u>.

E' il caso del numero composto 341=11*31, che risulta uno pseudoprimo su 120 basi rispetto a 340 dell'intervallo.

Altri esempi sono: 15=3*5 (solo 8 basi su 14), 91=7*13 (solo 48 basi su 90).

I numeri di Carmichel però, rispetto ai numeri primi, sono molto rari; mentre non è vero per i numeri pseudoprimi.

Ad esempio per il 15 la probabilità che non venga riconosciuto come composto è 8/14=0,57. Se si sottopone al test con un'altra base la probabilità che non venga riconosciuto come composto è 7/13=0,54, ancora ulteriormente è 6/12=0,5, 5/11=0,45,

4/10=0,4, 3/9=0,3, 2/8=0,25, 1/7=0,14. Quindi la probabilità dell'errore di non identificarlo come composto decresce.

Che probabilità c'è di individuare il 15, già per 4 volte non riconosciuto come composto, come composto al quinto tentativo? 0,57*0,54*0,5*0,45=0,07.

Di conseguenza se il controllo di primalità viene fatto per un numero grandi di basi es:100 la probabilità che non si individua che è composto ma un numero primo diventa:

$$\frac{1}{2^{100}} = \frac{1}{2^{10^{10}}} = \frac{1}{1024^{10}} \approx \frac{1}{10^{30}}$$

Una probabilità molto bassa, per cui un numero che dopo 100 tentativi non è riconosciuto come composto è al 99% un numero primo.

CRITTOGRAFIA

Per rendere crittografati dei caratteri vedremo che è più semplice utilizzare i numeri, il che consente una sostituzione rapida con un certo passo o secondo un algoritmo.

I numeri primi sono interessanti soprattutto perché non esistono attualmente teoremi tali da far recuperare in tempi brevi sia la chiave che il messaggio in chiaro (plain text); soprattutto si sfrutta la difficoltà di scomporre in fattori dei numeri primi molto grandi, ovvero con moltissime cifre.

Il giorno che si scopriranno i teoremi giusti, l'attuale crittografia crollerà come un colosso dai piedi d'argilla. Attualmente Il tutto è basato sull'aritmetica modulare e sui numeri naturali interi positivi primi.

La crittografia permette trasmissione di informazioni riservate per cui è adatta in vari ambiti: militare, polizia, bancario, trasmissione civile di dati sensibili e riservati, copyright di opere. Essa permette soprattutto la riservatezza e l'integrità del dato e abbinata ad altre tecniche permette l'autenticazione ed il non ripudio delle informazioni trasmesse.

CODICI A SOSTITUZIONE MONOALFABETICA CON CHIAVE SEGRETA

Uno dei primi codici cifrati fu il codice di Giulio Cesare, che costituisce un codice a sostituzione monoalfabetica.

Se facciamo riferimento alle 21 lettere dell'alfabeto italiano A = {a,b,c,?,z}, un testo T o messaggio in chiaro è una successione di caratteri di A.

Il codice cifrato di Giulio Cesare sostituiva il singolo carattere di T con un carattere presente in A ma a passo=3 da T e considerando A circolare.

Dire che A è circolare significa ad esempio che dopo z c'è c (passo=3).

Una tavola di sostituzione a passo 3 col codice di Giulio Cesare è:

a b c d e f g h i l m n o p q r s t u v z d e f g h i l m n o p q r s t u v z a b c

Il testo T={domani parto} si trasforma cifrato in C={grpdqn sduzr}.

Il codice quindi si basa sulla corrispondenza biunivoca, uno a uno, delle lettere.

In realtà si può generalizzare il codice basandosi sui concetti di
modulo (che indicheremo col simbolo @)
alfabeto considerato (21 lettere in italiano, 24 in inglese, e in cinese o in vietcong? Mah!).

Iniziamo a ragionare per l'italiano in modulo 21; per cui facendo una corrispondenza tra le 21 lettere dell'alfabeto e l'insieme Z20={0,1,2,3,...,20} si ottiene una tabellina del tipo:

a	b	c	d	e	f	g	h	i	l	m	n	o	p	q	r	s	t	u	v	z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

Se aggiungiamo anche un passo 3 si ottiene la tabella cifrata di seguito indicata:

a	b	c	d	e	f	g	h	i	l	m	n	o	p	q	r	s	t	u	v	z
3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	0	1	2

La tavola di può esprimere, in formula generalizzata. Se h è la cifra da crittare e p il valore di passo prescelto, allora la cifra crittata Cp(h) è:

$$Cp(h)=h+p \ @21$$

Il p in realtà è la "chiave segreta". Si può usare un valore di p compreso tra 1 e 20, ma p=0 non va usata, altrimenti il testo non risulta crittato!

Chi riceve il messaggio crittato c deve applicare il decrittaggio:

$$Dp(c) = c-p \ @mod \ 21$$

PUNTI DEBOLI

Il primo difetto individuabile è che lo spazio delle chiavi è piccolo; si è visto che p va scelto tra 1 e 21.

In altri termini la rottura del codice può avvenire provando le chiavi una dopo l'altra, con ricerca esaustiva (brute force) in un tempo relativamente breve.

Ma in quanto tempo si riesce a forzare un messaggio con una tecnica a forza bruta?

Facciamo un calcolo per il caso del codice monoalfabetico per alfabeto italiano: ci sono 21 lettere, per cui sono possibili 21! permutazioni da provare con attacco a forza bruta.

Ricordando che n! = n (n-1)(n-2) .... 2*1, per cui:

21!= 51.090.942.171.709.440.000 permutazioni

Se un calcolatore potente di tipo parallelo impiegasse solo 1 microsecondo (10-6 secondi) a permutazione, allora impiegherà circa un milione di anni per decifrare il tutto.

Apparentemente il codice è inattaccabile; tuttavia tutte le possibili permutazioni non portano a frasi di senso compiuto nell'ambito del linguaggio prescelto. Quindi come già visto è possibile far ricorso a tecniche statistiche di linguaggio (Attacco a dizionario).

In un linguaggio la probabilità che una determinata lettera si presenti con una certa frequenza è nota, per cui osservando nel messaggio crittato la frequenza con cui si presenta un carattere permette di ridurre drasticamente i tempi di decrittazione.

Per cui il codice di Giulio Cesare non è sicuro, ovvero è vulnerabile.

Il secondo difetto è che la chiave segreta è costituita da un solo carattere.

CODICI A SOSTITUZIONE POLIALFABETICA

Il codice è dovuto a Blaise de Vigenerè (1523-1596). La chiave in questo caso è una "parola".

Supponiamo che:

parola chiave = "voce", testo in chiaro = "se la sognano".

Si trasforma sia il testo che la parola chiave in una sequenza di numeri, partendo da zero e ripetendo la parola chiave n volte sotto il testo in chiaro per tutta la sua lunghezza w poi si sommano verticalmente messaggio e chiave ed il risultato, normalizzato in @21, dà il testo cifrato:

Testo	*S*	*E*	*L*	*A*	*S*	*O*	*G*	*N*	*A*	*N*	*O*
in chiaro	16	4	9	12	16	12	6	11	0	11	12
	V	*O*	*C*	*E*	*V*	*O*	*C*	*E*	*V*	*O*	*C*
Chiave	19	12	2	4	19	12	2	4	19	12	2
Testo	14	16	11	16	14	3	8	15	19	2	14
cifrato	Q	S	N	S	Q	D	I	R	V	C	Q

PUNTI DEBOLI

Si verificano i seguenti casi:

- Una stessa lettera in chiaro adesso può corrispondere a diverse lettere crittate. La "O" ad esempio è diventata una prima volta una "D" poi la seconda una "Q".
- Lettere diverse del testo in chiaro si possono trasformare in una stessa lettera crittata. Ad esempio sia "S" che "O" si trasformano in "Q".

Questo codice distrugge la possibilità dei metodi statistici basati sulla frequenza delle lettere nell'ambito delle parole del linguaggio. Anzi maggiore è la lunghezza e la casualità della parola chiave, più viene distrutta tale possibilità.

I codici di de Vigenerè sono, quindi, perfetti perché è praticamente impossibile risalire al testo in chiaro, ma hanno comunque dei punti deboli.

Se l'intercettatore conosce uno dei testi in chiaro: "se la sognano", può risalire dal testo crittato alla parola chiave. Difatti la parola chiave è ottenibile sottraendo in @21 il testo in chiaro da quello crittato.

Di conseguenza il codice di de Vigenerè va bene se si ha la cautela di cambiare ogni volta la parola chiave.

L'altro punto debole nasce se la parola chiave è più corta del messaggio in chiaro. Tale punto debole fu scoperto dall'ufficiale prussiano Friedrich Kasinski (1805-1881).

Il metodo è il seguente. Se la chiave è lunga m si possono formare m righe dove ad ogni riga si associa a turno una lettera del testo crittato. In questo modo si ottengono m codici con sostituzione monoalfabetica, attaccabili col metodo statistico.

Esempio. Chiave lunga 3 e testo lungo 1000 per cui devo creare tre righe:

Riga 1: c1c4c7…..c997c1000
Riga 2: c2c5c8…..c998
Riga 3: c3c6c9…..c999

Il metodo Kasinski si basa sul concetto che "coppie uguali di lettere che sono ad una distanza che è un multiplo della lunghezza della parola chiave, corrispondono a coppie uguali di lettere nel testo cifrato".
Oggi anche se la chiave fosse molto lunga esistono algoritmi su computer che possono risalire alla lunghezza della chiave.

Un algoritmo sicuro come AES (Advanced Encryption Standard) presuppone che la chiave cambi sempre. Il problema è, quindi, di far in modo, prima di attivare una comunicazione, di scambiarsi in modo sicuro una nuova chiave (crittografia a chiave privata).

La crittografia a chiave pubblica elimina questo tipo di problema.

LA CRITTOGRAFIA A CHIAVE PRIVATA

La crittografia studia come rendere un messaggio indecifrabile per poterlo far passare su un canale non sicuro. In realtà i concetti di crittografia sono poi applicabili ai protocolli per renderli canali sicuri.

Il crittoanalista è, invece, un analista di crittografia che da un testo cifrato deve riuscire a risalire al testo in chiaro.

La crittografia a chiave segreta presuppone che il testo in chiaro venga trasformato in crittato attraverso una chiave segreta o chiave di cifratura. Il destinatario disponendo di una chiave di decifratura (inversa della precedente) può dal testo cifrato risalire al testo in chiaro.

Questo vuol dire che le chiavi devono essere concordate tra il mittente ed il destinatario ovvero la chiave di decifratura è privata ed occorre passarla al destinatario su un canale che è generalmente insicuro. Questo è uno dei due motivi per cui difficilmente si usa una crittografia del genere.

Il secondo motivo è che, nei casi più semplici, tale tipo di crittografia presuppone di sostituire ogni simbolo del testo in chiaro con un altro (alfabeto permutato). Apparentemente, se si fa riferimento all'alfabeto inglese di 26 lettere, ci sarebbero un numero di combinazioni pari a 26! = $2^d\ 4*10^{26}$ e che con un "attacco a forza bruta" sarebbe impenetrabile, dato l'enorme tempo per provare tutte le combinazioni possibili. Difatti anche se si riuscisse a provare una chiave ogni microsecondo occorrerebbe un tempo pari a $13*10^{12}$ anni.

Sfortunatamente le parole con senso compiuto sono tali che ogni lettera di solito si presenta con una determinata frequenza nell'ambito del linguaggio prescelto. Per cui osservando la frequenza con cui si presenta un carattere crittato si può desumere abbastanza con precisione il tipo di lettera di cui si tratta. Basta disporre, rispetto al linguaggio, in gioco la tabella delle frequenze di ogni lettera.

PRINCIPIO DI BERONVSKY E PROBLEMI INTRATTABILI

In teoria "Qualsiasi testo crittato è possibile decifrarlo!" (Principio di Beronvsky) .

Il problema è solo il tempo ed i mezzi di elaborazione a disposizione.

Un problema difatti è definito "intrattabile" quando sono necessarie delle risorse (tempo o spazio) notevoli, al punto tale che l'applicare un attacco "a forza bruta" al testo non porta più nessun vantaggio (perché occorre un tempo notevole ad esempio, di molti anni).

Inoltre se "N" è la dimensione del problema da trattare il numero di operazioni da effettuare cresce come 2N. Tutto ciò è fonte di studio della Teoria della complessità computazionale.

Per cui se si lega un codice di crittazione ad un problema intrattabile, si riuscirà per anni (prima che i mezzi di elaborazione progrediscano) a rendere relativamente sicuri i messaggi scambiati su un canale insicuro.

CRITTOGRAFIA A CHIAVE PUBBLICA: RSA

Nel 1976 Martin E. Hellman ha proposto la crittografia a chiave pubblica. Il concetto su cui si basa la crittografia a chiave pubblica è che:

- si usano due chiavi: una pubblica da dare al destinatario ed una privata conservata dal mittente

- la chiave è fornita al destinatario pubblicamente su un canale insicuro
- l'algoritmo di crittazione è noto pubblicamente.

Un esempio di questa tipologia di crittazione sono gli algoritmi DSA, RSA etc.

Il sistema RSA fu proposto nel 1977 da Ronald L. Rivest, Adi Shamir e Leonard Adleman.

Gli algoritmi di successo normalmente sono quelli che legano la semplicità computazionale con il concetto di problema intrattabile. Ad esempio l'RSA cerca due numeri primi con un notevole numero di cifre e li lega al problema intrattabile di scomposizione del prodotto dei suddetti numeri primi.

Vediamo come funziona l'RSA e come se la caveranno i due ormai classici "Alice e Bob" a scambiarsi un messaggio segreto.

Alice crea due chiavi: una pubblica ed una privata. La chiave pubblica la dà a Bob o anche ad altri amici e la privata la tiene per sé.

Come sono fatte la chiave privata e quella pubblica?

La **chiave pubblica** è creata cercando casualmente due numeri dispari p e q con un centinaio di cifre e che al test del "Piccolo teorema di Fermat", facendo la verifica su almeno 100 basi, risultano primi.

Se p e q risultano primi, allora si sceglie:

m=pq e N(m) = (p-1)(q-1)

e si sceglie un E primo rispetto a N(m).

A questo punto si ottiene la chiave pubblica:

C= PE@m

Per la chiave privata occorre determinare D in modo che:

DE@N(m) = 1

Da cui si ricava che la chiave privata è:

P = CD@m

Come potrebbe un crittoanalista, a partire dalla chiave pubblica, risalire alla chiave privata?

In pratica è un problema intrattabile, perché occorre fattorizzare un numero a 200 cifre ovvero m, poi si deve calcolare N(m) e ricavare D. Se si tenta di estrapolare, da un insieme limitato di valori di C, il valore P è impossibile, perché come visto non esiste una legge regolare (C è ad andamento disordinato) con cui ricavare tutti i valori di P.

In realtà ad un numero m di 200 cifre corrispondono un numero di resti elevato tra:

10199 e 10200 -1

il che è anche una protezione eccessiva. E' sufficiente un frazionamento del numero P in blocchi di ugual cifre ad esempio con blocchi Pi con 11 cifre. Il numero di resti è comunque compreso tra 1010 (10 miliardi) e 1011 -1 (100 miliardi -1).

ESAMINIAMO MEGLIO IL PROBLEMA

Esaminiamo meglio il problema per produrre un software didattico con PARI/GP.

Come è noto la crittografia del RSA è detta asimmetrica perchè fa uso di due chiavi: una pubblica ed una privata; mentre la crittografia simmetrica ha una sola chiave da tener segreta.

Adesso ci poniamo il problema di come crittografare in un algoritmo un messaggio in RSA.

La teoria è nota; i passi sono:

1. Alice sceglie casualmente, col suo algoritmo, due numeri primi p e q distinti e grandi (circa trecento cifre) e calcola N = p*q;
2. si calcola la funzione di Eulero phi(N)= (p -1)(q -1);
3. l'algoritmo "butta" via p e q (sono da tenere segreti);
4. si determina la coppia KPu=(puK,N) che costituisce la chiave pubblica che Alice darà agli amici (es: Bob), che la useranno per inviare messaggi ad Alice;
5. Alice si conserva la chiave privata KPr=(N,prK);
6. puK e prK devono esser ricavate in modo da rispettare determinate regole: puK e phi(N); devono essere coprimi e puk * prK deve essere congruo 1 mod phi(N);
7. L'algoritmo di Alice a questo punto può fare a meno anche di phi(N);

Ora Bob può inviare un messaggio ad Alice usando la chiave publica che Alice ha distribuito a tutti; mentre Alice potrà decrittare i messaggi con la chiave privata che deve conservare gelosamente.

Se Bob volesse ricevere anche lui messaggi sicuri dai suoi amici dovrà creare a sua volta una coppia di chiavi proprie: quella pubblica da dare agli amici e quella privata da conservare gelosamente.

La generazione delle coppie di chiavi è fatta una sola volta.

Che c'entra la fattorizzazione? C'entra!

Alice a Bob dà N e puK ma Bob sa soltanto che N che è un composto (semiprimo RSA) con moltissime cifre e dispone del puK. Se riuscisse a fattorizzare N in tempi brevi saprebbe i due numeri primi p e q, si calcolerebbe phi(N) e di conseguenza sarebbe capace di calcolarsi prK.

Per cui i messaggi non sarebbero più segreti!

Qual è il punto di forza/punto debole del RSA? Il problema è che "fattorizzare N" è un problema non polinomiale e in generale P è diverso da NP; il che vuol dire conosciamo bene un metodo deterministico (un algoritmo) per fattorizzare N ma per poterlo completare servono tempi tanto più elevati quanto più è la lunghezza del numero di cifre di N.

Basta una chiave a soli 1024 bit ed è inutile solo tentare.

In appendice proponiamo un algoritmo didattico di crittografia RSA e dei problemini che pone.

Il messaggio di partenza è convertito in numerico ottenuto dalla somma di ogni valore del carattere, nell'alfabeto scelto (A...Z di 26 caratteri), moltiplicato per la potenza di 27 della posizione del carattere nel messaggio. Qui si è scelto di partire dall'esponente 1 della potenza.

Il blocco di test conveniente da crittografare è dato da log n / log 26 circa 22 caratteri. Per cui se il testo è maggiore va suddiviso in blocchi e crittografato: ad esempio ogni blocco costituisce una riga scritta in un file. L'algoritmo didattico non prevede lettura e scrittura da file ma è facilmente estendibile anche in tal senso.

La decrittazione usa la funzione inversa del crypting. La conversione da numero del messaggio decrittato a stringa avviene con una conversione basata su potenze di 27.

Perchè potenza di 27 e non di 26? una comodità algoritmica (è "un consentitemi di ..." informatico!). Il problema era Z, corrisponde al valore 90-64=26, e quindi si sarebbe ottenuto nell'algoritmo di seguito un valore di 26*26^weigth e nella decodifica, da numero a testo, avremmo avuto un errore perchè la max potenza cercata sarebbe stata maggiore di una unità rispetto al valore effettivo. Invece con 26*27^weigth i conti tornano rapidamente e si semplifica il numero di linee di codice necessarie. E' questo il motivo che spesso al primo carattere A dell'alfabeto si dà valore 0 mentre a Z e l'ultimo 25. Solo che gli array in PARI/GP partono da 1 ed era scomodo.

Con un pò di lavoro si può estendere l'esempio anche ad un alfabeto alf>26 caratteri, contenente anche spazi, numeri, punteggiatura, lettere minuscole, parentesi etc, ovviamente modificando il software con potenza alf. Nell'esempio vanno invece usate solo lettere maiuscole senza spazi etc.

Esempio di messaggio da crittare: ANTONIOVAACASA

ESEMPIO CHIAVI PRODOTTE CON RSA E FRAZIONAMENTO DI P

Useremo dei valori p e q di esempio, che per facilità di calcolo saranno valori molto bassi (2 cifre) e quindi poco sicuri.

Nel seguito il discorso è puramente didattico per comprendere come si lavora col frazionamento di p nell'algoritmo RSA. Nella realtà occorre scegliere randomicamente valori di p e q dispari e primi, con almeno 100 cifre.

Alice ha generato due numeri dispari di sole due cifre per semplicità p=47 e q=61. Di conseguenza è:

m=pq=47*61=2867, N(m)= (p-1)(q-1)=2760, si sceglie E=1183, per cui è:

$$C=P^{1183}@2867$$

$$P=C^{7}@2867$$

Bob vuole trasmettere ad Alice la frase: “Sono uscito” e vuole usare la sua chiave pubblica C.

Si può utilizzare il codice ASCII per codificare il tutto in cifre numeriche per ogni carattere del messaggio in chiaro, sia minuscolo che maiuscolo sia numerico che caratteri speciali e punteggiatura.

Ad esempio: S corrisponde a 44, o invece corrisponde a 14 etc.; per cui il messaggio in chiaro diventa ad esempio:

P=4414131462201802081914

A questo punto P è frazionabile in blocchi di uguali dimensione ad esempio 3; per cui è:

P1=441
P2=*413*
P3=146
P4=220
P5=180
P6=208
P7=191
P8=462

L'ultimo blocco P8 è stato aggiunto il blank corrispondente a 62 in ASCII per far in modo di avere sempre 3 caratteri a blocco.

A questo punto Bob può determinare gli 8 blocchi cifrati Ci:

$C_1=441^{1183}$@2867=2515
$C_2=413^{1183}$@2867=1572
$C_3=146^{1183}$@2867=1868
$C_4=220^{1183}$@2867=1426
$C_5=180^{1183}$@2867=1376
$C_6=208^{1183}$@2867=1338
$C_7=191^{1183}$@2867=1427
$C_8=462^{1183}$@2867=295

Cosa deve fare Alice adesso che riceve gli 8 blocchi Ci?

Deve usare la sua chiave privata P, con la quale ricava ogni Pi=Ci7@2867, dopo di che se uno dei valori è di numero inferiore alle tre cifre deve aggiungere degli 0 a sinistra, ricomponendo P, poi decodifica P entrando nella tabella del codice ASCII col valore ricavando il carattere corrispondente risalendo al testo in chiaro.

PRO E CONTRO DEL RSA

La sicurezza del RSA è, quindi, basata sul concetto che per violarlo occorrerebbe individuare i fattori primi della chiave pubblica per decifrare il messaggio. Ma la fattorizzazione di numeri primi enormi (oggi si usano chiavi pubbliche a 1024 bit o 2048 bit in campo militare) richiede tempi proibitivi.

Il difetto maggiore del RSA è la mole di calcoli da fare per numeri primi elevati che si traduce in una certa lentezza (specie per l'elevamento a potenza per cui occorrono algoritmi ottimizzati). Ad esempio l'RSA è sicuramente 1000 volte più lento del DSA. Tuttavia per i computer moderni tale problema è accettabile vista la potenza hw in gioco.

Goldbach e la fattorizzazione dei semiprimi e del RSA

Se si utilizza una equazione di secondo grado e la congettura forte di Goldbach si trova un semplice metodo per fattorizzare i semiprimi RSA; metodo individuato da James Constant (math@coolissues.com).

Il problema di Goldbach come abbiamo visto è connesso a Riemann e mostreremo che non avremo bisogno della funzione totiente φ(x) di Eulero e che è possibile fattorizzare un semiprimo N a partire solo da N.

Siano a e b due numeri primi, che non conosciamo a priori, allora otteniamo il semiprimo:

p=a*b (98)

Poniamo

a+b=s (99).

Ora per la congettura di Goldbach se a e b sono primi allora s è un numero pari.

Poniamo:

b=s-a (100)

Dalla (98) si ottiene l'equazione di secondo grado:

$$a^2 - a\,s + p = 0 \quad (101)$$

Da cui è:

$$a = \frac{1}{2}(s \pm c) \qquad c = \sqrt{s^2 - 4p} \quad (102)$$

Ora se a e b sono primi ed s pari, poiché $2a = (s \pm c)$ allora anche c è pari!!!

Ora se c è pari e c >0 significa che

s2 > 4p (103)

Deve quindi essere:

s2 = c2 + 4p (104)
c2 = s2 - 4p (105)

Metodo algoritmico per fattorizzare in base a Goldbach

- Step 1: si cerca un valore di $s^2 > 4p$ tale che sia un quadrato
- Step 2: si determina $c^2 = s^2 - 4p$.
- Si estrae la radice quadrata di c^2. Se c risultante è intero e numero pari ci si arresta perché con la (102) e la (100) si sono trovati a e b
- Altrimenti si ritorna allo step 1 per il quadrato successivo

Esempio

p=55
4p=220

Un quadrato successivo è $s^2=16^2= 256$

Ora dalla (101) è $c^2 = 256 - 220 = 36$, da cui c=6 è pari e intero.

Dalla (102) è a = ½(16±6) ovvero le due soluzioni a=11, b=5. Soluzione trovata e Fattorizzazione effettuata!

E' generalizzabile il metodo ad un numero di fattori qualsiasi?

Sì, ma con qualche problema come vedremo.

La generalizzazione del metodo significa poter fattorizzare un numero $N=p_1p_2p_3p_4\ldots p_m$ di cui non conosciamo a priori né *m, che chiamiamo grado di fattorizzazione o bigomega*, né i vari p_i.

Inoltre si dovrebbe aggiungere una equazione sugli m fattori primi, attraverso una congettura di Goldbach o una sua variante (*Vinogradov*) o qualche altra proprietà che coinvolge gli m fattori, sia nel caso di risultato pari o dispari. In tal caso si otterrebbe una equazione di grado m qualsiasi (da non risolvere), i cui coefficienti dell'equazione però devono sottostare a m-1 disequazioni. Le disuguaglianze su cui basano le disequazioni sono dedotte a partire dalla somma dei fattori. La soluzione del sistema di disequazioni fornisce alla fine soluzioni che rappresentano i fattori primi di N.

Il problema è risolvibile se, nell'ordine, rimuoviamo due difficoltà:

- esiste un modo per sapere a priori dato N, quanti m fattori ha N ivi comprese le molteplicità (ad esempio se $N=2*3^2*5^3$ allora m è la somma degli esponenti di tutti i numeri, m=6) ?
- possiamo usare Goldbach o varianti con la somma di m fattori per avere delle condizioni da imporre nel sistema di disequazioni?

Il primo punto non è impossibile, fidatevi! Se siete avvezzi al programma PARI/GP e alla Teoria dei numeri saprete che esiste una *funzione bigomega(x)*, molto veloce, che fornisce il numero di divisori di x, tenendo conto anche della loro molteplicità.

Ad esempio N=90:
m = bigomega(90)=4, perché $90=2*3^2*5$

inoltre anche con numeri con molte cifre si ottiene una risposta in pochi millisecondi.
Dov'è l'inconveniente in tutto ciò? Se ad esempio m=200 avremo un'equazione di grado 200 e quindi da risolvere un sistema di 199 disequazioni (questo non è un problema per un calcolatore): il problema sono le ipotesi sulle 199 disequazioni da formulare e generalizzare per ogni m. In teoria è possibile. Conviene tale metodo in pratica? Per l'RSA si usano i metodi economici visti prima, per la fattorizzazione basta scomporre in fattori… Forse la generalizzazione del metodo, al momento, non

ha un’applicazione per cui convenga usarlo. Ma ce lo segniamo sul Block Notes Matematico, non si sa mai in futuro!

Fattorizzazione e quadrati perfetti

L’algoritmo del quadrato perfetto permette di fattorizzare istantaneamente il prodotto di due numeri primi gemelli p e q, poiché, per motivi legati alle forme $6k\pm1$, tali prodotti sono di forma $p * q = (6k-1)(6k+1) = N$ e in tali casi:

$$p = \sqrt{(N+1)} - 1 \quad q = \sqrt{(N+1)} + 1$$

dove 1 è il quadrato della semidifferenza tra due numeri gemelli; per numeri primi non gemelli, ovviamente la semidifferenza è maggiore e quindi anche il suo quadrato;

un esempio per tutti $p = 11$, $q = 13$ (primi gemelli)

$$N = p * q = 11*13 = 143$$

$$p = \sqrt{(143+1)} - 1 = \sqrt{144} - 1 = 12 - 1 = 11$$

$$q = \sqrt{(143+1)} + 1 = \sqrt{144} + 1 = 12 + 1 = 13,$$

Quindi vanno evitati i primi gemelli.

Fattorizzazione dei semiprimi e del RSA col modulo

Una fattorizzazione basata sull’equazione $x^2=a^2 \bmod N$ dove $N=p*q$ è il semiprimo, quindi, dello stesso genere RSA, è mostrata in APPENDICE come variante del metodo di Fermat.

AUTENTICAZIONE E RISERVATEZZA

L’autenticazione è quel processo che garantisce l’identità del mittente del messaggio; mentre la riservatezza è la capacità di far sì che il messaggio trasmesso non venga compromesso e che sia fruibile solo la destinatario possessore della chiave.

La crittografia a chiave pubblica e l’algoritmo ad esempio RSA consentono anche tali requisiti.

Se Bob vuole dimostrare a tutte le persone a cui ha fornito la sua chiave pubblica che un doc è effettivamente stato scritto da lui (autenticazione) deve crittare il documento con la sua chiave privata; in sintesi: B.pr(doc)

Se Bob vuole dimostrare solo e soltanto ad Alice che un documento è stato scritto da lui e non è stato manipolato (riservatezza) è sufficiente che lo sottopone a crittografia con la propria chiave privata e poi il tutto con la chiave pubblica data ad Alice.

In sintesi: A.pu(B.pr(doc)).

Per ricostruire il doc, Alice deve compiere le stesse operazioni inverse di Bob. In pratica Alice deve usare prima la propria chiave privata e poi quella pubblica di Bob.

La scelta di Bob di crittare nell'ordine A.pu(B.pr(doc)) oppure B.pr(A.pu(doc)) è solo data dal vincolo di iniziare dalla chiave col modulo minore.

Crittografia: curve ellittiche - il logaritmo discreto - algoritmo di El Gamal

Nella crittografia asimmetrica, ovvero a chiave pubblica e privata, esistono anche altri algoritmi interessanti, basati ad esempio sulla teoria delle curve ellittiche e che fanno leva sulla grossa difficoltà di risolvere in tempi brevi un "problema di fattorizzazione discreta" noto anche come problema del logaritmo discreto.

Qui addirittura le chiavi pubbliche scambiate tra Alice e Bob, come vedremo, hanno anche maggiori informazioni rispetto ad una tecnica RSA.

Supponiamo che Alice compia i seguenti passi:

1. sceglie randomicamente un numero primo $p>1$ a molte cifre, poi un intero g compreso tra 1 e p ed un intero a compreso tra 1 e p.
2. calcola il valore $g^a mod p$
3. crea la propria chiave pubblica $KpuA=(p, g, g^a)$ e la da ai suoi amici (es: Bob)
4. la propria chiave privata è Kpr=a

Bob, ricevuta KpuA da Alice farà i seguenti passi:

1. genera un proprio intero b: b appartenga a Zp (ovvero tra 1 e p)
2. costruisce la propria chiave pubblica $KpuB=(p, g, g^b)$ e la da ad Alice
3. custodisce la propria chiave privata KprB=b
4. sceglie un messaggio M apaprtenente a Zp

Bob adesso cripta M e lo trasmette ad Alice. Bob esegue il crypting nel seguente modo:
$Mc = M(g^a)^b mod\ p$

Alice conosce KpuB, quindi g^b.

Da qui calcola $(g^b)^a mod\ p$

A questo punto Alice è in grado di decriptare il messaggio Mc perchè:
$Mc(g^b)^a)^{(-1)}=M(g^a)((g^b)^a)^{-1}=M$

Logaritmo discreto e Fattorizzazione discreta

Cosa c'entra il logaritmo e perchè è discreto?

Alice conosce la chiave pubblica di Bob in cui c'è p e g^b o meglio g^b mod p; per cui per sapere la chiave privata di Bob dovrebbe ricavarsi b da g^b mod p. In pratica g è la base b l'esponente dell'elevazione a potenza. L'operazione inversa di ricavare l'esponente (b) a cui elevare la base (g) per ottenere l'argomento è il logaritmo! Solo che lavoriamo in aritmetica modulo ecco perchè si parla di logaritmo discreto.

Ebbene l'operazione di ricavarsi il logaritmo discreto tenendo conto che sia g che b sono a molte cifre è banale ma richiede molto tempo. E' questo il punto di forza di questa crittografia.

Un algoritmo noto per affrontare il logaritmo discreto è simile a quello di Fermat.

Curve ellittiche

Cosa c'entrano le curve ellittiche? Studiando le curve ellittiche e, ad esempio il problema del millennio di Birch, Swinnerton e Dyer, si scopre che, per poter trattare l'infinito numero di punti a valore razionale Q, è conveniente far riferimento alla aritmetica modulo. Ma questo lo affronteremo un'altra volta e con calma.

Software didattico

Segue un esempio di algoritmo didattico su EL GAMAL in appendice, dove alcune funzioni sono comuni con il blog precedente del RSA e che troverete lì facilmente. La libreria invece è disponibile su INTERNET grazie all'autore.

STEGANOGRAFIA

Se la crittografia ha come obiettivo di rendere i messaggi riservati, la steganografia interpreta tale esigenza in modo del tutto creativo.

I concetti di essa risalgono a Johannes Trithemius (1462-1516) che li introdusse nel suo trattato "Poligraphie Libri Sex". In tale trattato venivano discussi principalmente i codici a sostituzione poli-alfabetica come quelli di de Vigenerè, utilizzati tra l'altro anche nella seconda guerra mondiale dai tedeschi con la famosa macchina Enigma.

Thrithemius era il nome "italianizzato" di Johannes von Heidenberg, monaco benedettino, abate a Sponheim, dove fondò una biblioteca passata successivamente al Vaticano. L'abate scrisse molti libri di varia natura: teologici, alchemici, di medicina ed altro. Nel 1499-1500 scrisse il libro Steganographie, che circolò solo nel 1606, perché iscritto tra i libri pericolosi, in quanto considerato pieno di superstizioni, astrologia e assurdità (angeli che trasportano messaggi). Tra l'altro la prefazione del libro annunciava la presenza di un testo cifrato nel libro. Solo nel 1998, John Reeds della AT & T è riuscito a trovare sia la chiave nascosta che il messaggio. Si tratta di un testo però confuso come se si fosse perso qualche pezzo.

Oggi la steganografia viene usata anche per i copyright. Si tratta di utilizzare una immagine e introdurre in essa un messaggio (o una firma per il copyright), cambiando il colore di una piccola percentuale di pixel così da non rendere evidente la modifica.

Un software di pubblico dominio che permette di fare steganografia su una immagine jpg si chiama steghide.

Se si vuole nascondere un messaggio nella jpg di nome ad esempio immagine.jpg, basta utilizzare un file di testo segreto.txt contenente il messaggio e fare da DOS:

steghide embed -cf immagine.jpg -ef segreto.txt

Viene chiesta una password o chiave segreta privata, che va condivisa con chi riceve l'immagine. Il comando per decifrare è:

steghide estract -sf immagine.jpg

Anche qui va introdotta la chiave privata. Alla fine ci si trova il file segreti.txt col contenuto del messaggio.

APPENDICE

```
/*
*  Cryptography Area
*
*/
/**********************************************************
* Am example of Crypt/Decrypt in RSA with public key e private key
*
* Out alphabet is of 26 char:
* A, B, C, D, E, F, G, H, I, J, K, L,M, N, O, P, Q, R, S, T, U, V,W, X, Y, Z
*
* Example of Word :
* HELLOWORLDBYROSARIOTURCO
*
* Attention: max digits of text you can crypt with private key must be:
* log n / log 26
* Text of 20 chars for time
*
* Example of use in sequence:

  createCryptRSA()              [here -> KPr=(N, prk) KPu=(N,puk)]
  get26Ascii("HELLOWORLDROSARIOTURCO")   [here -> m ]
  cryptWordRSA(m, N, prK)              [here -> secret]
  decryptWordRSA(N, puK, secret)

  or
  testRSA("HELLOWORLD")
*
*
*
* Rosario Turco
**********************************************************/
```

```
{createCryptRSA() = local (p,q, KP, phi);

 KP=vector(3);

 p = nextprime(random(10^40));

 q = nextprime(random(10^35));


 KP[1]=p*q;             /* N */

 phi = (p-1) * (q-1);  /* phi N */

 print("phi : ", phi);


 /*
 *  Person A produce own private key and public key */
 /* Public key = (N, puk) */
 /* Private key = (N, prk) */

 /* I put them in a vector KP for convenience in the test*/

 puK = random(q);

 print("\nN   : ", KP[1]);

 while(gcd(phi,puK)!=1,puK=puK+1);

 KP[2]=puK;
 print("puK : ", puK);


 prK = lift(Mod(puK,phi)^(-1));


 KP[3]=prK;
 print("prK : ", prK);


 /* I verify that prk*puk is congrue con 1 mod phi */
 test = (prK * puK) % phi;

 if( test == 0,
    error("Test NOK: It isn't congrue 1 mod phi"); );

 print(" ");

 return(KP);
```

```
}

/*
*  External Function
*/

{get26Ascii(str) = local (MyW, weight, i, a, m);


 weight = length(str);

 MyW=Vecsmall(str);


 m=0;

 /*
   I use a power of 27 otherwise char Z will have
   26*26^weight and this doesn't work well
   with getMsg
 */

 for( i=1,weight,

     a = MyW[i]-65+1;
     m = a*(27^weight) + m;
     weight--;
  );


 return(m);

}


/*
*   External function
*/
{getMsg(num) = local (a=num,d=27,r=0, msgM="", amax=0);

 print("num :", num);

 print("d :", d);
 while( d>26,

       amax = maxk26k(a);
       d = floor(a/(27^amax));
       r = a%(27^amax);
       msgM=concat(msgM,Strchr(d+64));

       if( r>=26, a = r; d=27;);
```

```
        if( r<26 & r!=0, msg=concat(msgM,Strchr(r+64)); d=1;);

      );

  return(msgM);

}

/*
*  Internal function
*/

{maxk26k(n) = local (k);

 k=0;

 while( 27^k <= n, k++;);

 return(k-1);

}

/*
*   External function
*/

{cryptWordRSA(m, N, puK) = local (secret);


 secret = lift(Mod(m,N)^puK);
 print("\nsecret value: ", secret);
 print(" ");


 return(secret);


}
/*
*   External function
*/

{decryptWordRSA(secret, N, prK) = local (desecret, msg);

 desecret =  lift(Mod(secret,N)^prK);

 print("decrypt value: ", desecret);

 print ("\nTest OK if decrypt value is equal to m value");

 return(desecret);
```

```
}

/ *
   Test function

   Person B transmit a message to A with public key di A
   Person A decrypt with own private key

   If person A transmit a message to B must use the public key of B



*/

{testRSA(strMyWord) = local (vec,ms,secretRSA,desecretRSA,msg);

 /* B has got a KeyPub and transmit to A  a message */
 Key=vector(2);
 Key=createCryptRSA();

 ms = get26Ascii(strMyWord);
 print("m   : ", ms);

 secretRSA=cryptWordRSA(ms, Key[1], Key[2]);


 /* Now A decrypt */

 desecretRSA=decryptWordRSA(secretRSA,Key[1],Key[3]);

 msg = getMsg(desecretRSA);
 print("\nmsg decriptato: ", msg);


 return;

}

/*
* EL GAMAL
*
* External function
*
*/

{createCryptElG() = local (KPEG, p, g, a, b, ga, gb);

KPEG=vector(6); /* It's Public Key of El Gamal */
```

```
p = nextprime(random(10^40));

g = random(p);

a = random(p); /* It's Private Key of El Gamal user A */

b = random(p); /* It's Private Key of El Gamal user B */

ga=lift(Mod(g,p)^a);
gb=lift(Mod(g,p)^b);


/* I put them in a vector KPEG for convenience in the test*/

KPEG[1]=g;
KPEG[2]=a; /* a for user A and b for user B */
KPEG[3]=ga; /* ga for user A and gb for user B */
KPEG[4]=p; /* p is shared */
KPEG[5]=b;
KPEG[6]=gb;


return(KPEG);

}

/*
* External function
*/

{cryptWordElG(mc, p, a, gb) = local (msecret);


msecret = lift(mc*Mod(gb,p)^a);
print("\nsecret value: ", msecret);
print(" ");


return(msecret);


}

/*
* External function
*/

{decryptWordElG(secretElG, p, b, ga) = local (desecretElG, msg);
```

```
desecretElG = lift(secretElG*(Mod(ga,p)^b)^-(1));

print("decrypt value: ", desecretElG);

return(desecretElG);


}

{testElG(strMyWordTest) = local (msc,Msecret,Mdesecret,msgs);

/* B has got a KeyPub and transmit to A a message */
Key=vector(6);

Key=createCryptElG();

/* B tx to A */
msc = get26Ascii(strMyWordTest);
print("msc : ", msc);


Msecret=cryptWordElG(msc, Key[4], Key[2],Key[6]);

/* A decrypts */
Mdesecret=decryptWordElG(Msecret,Key[4],Key[5],Key[3]);

print("Decodifica\n");
msgs = getMsg(Mdesecret);
print("\nmsg decriptato: ", msgs);


return;


}

/*
*   crackRSA(n)
*
*   It prints two prime factors of a sempiprime RSA
*   such that n=p*q
*
*   It uses few lines of code
*
*   Use:
*
*   crackRSA(getRSA(1))
```

```
*   or
*   crackRSA(n)
*
*   It uses a strategy MPQS
*
*
*   Note: If you use with getRSA(1,n) we raccomend
*   n max 25 digit, otherwise you obtain the message
*   floor: precision too low in truncr (precision loss in truncation).
*   in crackRSA
*
*   This is a didactic software.
*   Function factor in PARI is faster.
*
*/

{crackRSA(p) = local (s, vec, qp);

  vec=vector(2);

  qp = 4*p;

  s=floor(sqrt(qp));

  while( !(s^2>=qp) | (issquare(s^2 - qp) == 0) | (frac(s^2 - qp)!= 0.0),
        s++;
       );

  vec[1]=floor((s+sqrt(s^2-qp))/2);
  vec[2]=floor((s-sqrt(s^2-qp))/2);

  return(vec);
}

/*
*  If it returns [0], is a prime number.
*  scompFactor uses bigomega
*
*  This is a didactic software.
*  Function factor in PARI is faster.
*
*/
{scompFactor(p) = local ( col, f1, f2, f3, s, i, jOut, vecIn, vecOut, vecStack);

  vecIn=vector(2);

  col = bigomega(p);

  vecOut=vector(col);
  vecStack=vector(2*col);
```

```
  print("\nscompFactor");
  print("\n#fattori previsti: ", col);

  f1=0;
  f2=0;
  f3=0;

  s=1;
  jOut=1;
  iStack=1;

  while( s<col,

        vecIn=crackRSA(p);

        if( isprime(vecIn[2]), vecOut[jOut]=vecIn[2]; jOut++; f1=1; print("->",vecIn[2]););
        if( isprime(vecIn[1]), vecOut[jOut]=vecIn[1]; jOut++; f2=1; print("->",vecIn[1]););


        if( s<col & f2==1 & f1==0, p=vecIn[2];);
        if( s<col & f1==1 & f2==0, p=vecIn[1];);

        if( s<col & f1==0 & f2==0, vecStack[iStack]=vecIn[1];
           p=vecIn[2]; iStack++; vecIn[1]=0; vecIn[2]=0; f3++;);

        if( iStack>1 & s>1 & f1==1 & f2==1 & (vecStack[iStack-1]!= 0),
            p=vecStack[iStack-1]; f3--; vecStack[iStack-1]=0; iStack--;
        );

        if( iStack==1 & s>1 & f1==1 & f2==1 & (vecStack[iStack]!= 0), p=vecStack[iStack];
f3--;
            vecStack[iStack]=0;
        );


        if(p==1, s=col;);

        f1=0;
       f2=0;
        s++;

       );

  return(vecOut);
}

/*
*******************************************************
* Quadratic Sieve Factoring
*
*     x^2 = a^2 mod N
```

```
*     x^2 = a^2 + k*N (k=0, k=1)
*
* Variant of Fermat's algorithm
* This method is good for odd numbers
*
* Rosario Turco
*****************************************************
*/

{qsfRSA(n) = local (col, vecOut, xq, a, x1, x2);

 if( n<2 | frac(n) != 0.0 , error("qsfRSA(n): You must insert an integer n>1"));

 col = bigomega(n);

 /* trivial solution: 1 and n it is a prime*/
 if( col == 1, print1("qsfRSA: It's a prime number: ", n); return(););

 vecOut=vector(col);

 if( col != 2, print1("qsfRSA: It isn't a RSA number. You must insert a RSA number");
return(););

 /* if it is a even number ... */
 if( Mod(n,2)==0, vecOut[1]=2; vecOut[2]=floor(n/2); return(vecOut););

 xq=0;  /* xq = x^2 */
 a=0;

 while( (!ispower(xq,2) | xq == 0), a++; xq=a^2+n;
 );

 x1=a;
 x2=floor(sqrt(a^2+n));

 vecOut[1]=x2-x1;
 vecOut[2]=x2+x1;

 /* if n is a perfect square */
 if( x2-x1==1 & x2+x1==n, vecOut[1]=floor(sqrt(n)); vecOut[2]=vecOut[1];);

 return(vecOut);

}
```